Coursework: Large-Scale Text Classification

This coursework has been completed as per Coursework Version 1

Question 1:

Text-Part Algorithms:

Local Version:

./bin/spark-submit --driver-memory 5g /home/dan/Desktop/IMN432-CW01/ackf415-new.py > /home/dan/Desktop/IMN432-CW01/output.txt

Lewes Version:

/usr/local/spark/bin/spark-submit --driver-memory 12g ackf415-lewes.py > output.txt

Optimising Logistic Regression Version:

./bin/spark-submit --driver-memory 5g /home/dan/Desktop/IMN432-CW01/ackf415-new-optimisation.py > /home/dan/Desktop/IMN432-CW01/output.txt

Text-Full Algorithm:

Lewes Version: Execution of the Text Classification algorithm can in be initiated using the following command:

/usr/local/spark/bin/spark-submit --driver-memory 25g ackf415-lewes-textfull.py > output_TF.txt

Element 1: The command initiates Apache Spark local on your machine.
Element 2: The command, when declared, initiates Spark with an allotted amount of memory. In this case 5 GB.
Element 3: The location of the Python file where the algorithm is located
Element 4: This is the output file, one location where all the results and errors (when developing) would be printed.

Each of the algorithms have two additional configurations; firstly a setting for defining the allocation of the virtual memory and secondly the number of cores to use (lines 475-478). I only tested this algorithm on my own machine, I made full use of the 4 cores and hyper-threading capability; this equates to 8 available cores available for processing.

The table below should assist with identifying each part of the question;

Question	Task	Line Range
1a	Load all the text files using TextFile()	45-67 – Directory Traversing Function
		480-500 – Loading each file in the directory list and processing
		171-236 – Processing Function for each file
1b	Removal of the Header	182-190 – Dealing with the Header of Each file
		101-114 - Function for identify the line which the header finishes at
	Removal of the Footer	115-127 – Dealing with Footer of each file
		191-199 – Function for identifying the line where the footer starts on
		206-208 – Removing the header and footer to get a subset of the text
1c	Extracting the Ebook Number	200-203 – Finding the Ebook number of each file
		131-168 – Function for find the Ebook number
1d	Extract Word Frequency pairs per file and save RDD to disk	215-235 – Extracting the Word Frequencies and saving to a Pickle File
1e	Calculation of the IDF Values and save RDD to disk	501-523 – Processing of the IDF Values
1f	Calculation of the TF.IDF vectors	525-547 – Calculation of TF.IDF Vector Spaces
		319-333 – Creating Hashed Vector Spaces

Detailed Explanation of each Stage in the Algorithm:

Question 1a

When commencing with this coursework, the decision was made to use the textFile method instead of batch processing multiple files in operation, this meant that the processing of the word frequencies would be significantly slower than the alternative method when scaling up to the full dataset. When importing the file into a Resilient Distributed Dataset (RDD) an index was combined as well as the lower casing of every feature. The purpose of the index was to easily subset the data and to exclude both the header and footer between calculated values. Using the directory function, Spark will pre-process each file in the directory list before continuing on.

The decision was made to only process files that are less than 1MB in size. Including files which are of greater size than this would dramatically impact the performance of Spark; some of the excluded files reached 50MB in size. I am aware that file sizes in industry would likely exceed this size however for the purpose of this coursework I chose to remove them, a cluster would have significantly greater computational power to process these filesizes than my laptop. I have also chosen to exclude all the files that contain a hyphen in the file name. The purpose of this was to eliminate duplicated and revised entries from the dataset.

Question 1b

Removing the header and footer was a trial and error process to identify all of the possible combinations of the headers. From experimenting with the data I was able to extract three prevalent types for both the header and footer, as shown on lines 103-105. The method used captures approximately 95% of the headers in the Text-Part dataset. For both the header and footer, a value of the index was obtained; where no value was found, this would be set as the first line or the last line of the header and footer respectively. The subset was extracted using the values for the previous two functions using the index.

One concern I have with this method is that I have made the assumption that prior to distributing to the multiple cores on my own machine the master node in Spark attaches an index. I tested this and there appears to be no impact to the text and, when collated, does in fact have an element of the header or footer.

Question 1c

Using a recursive process I found that there were three main ways to identify how the eBook number is stored within the textFile. Again, this method has a high accuracy rate. The function makes full use of regular expressions to determine where in the file the eBook number is extracted from.

Question 1d

The final sub-stage of processing the file was to find the Word Frequencies and save to a pickle file. Prior to saving to pickle file, the data was repartitioned to reduce the number of pickles in each of the files word frequency directories. Before saving the RDD, the maximum frequency was calculated and each of the summations of the words was divided by the maximum frequency to acquire the word frequency vector.

Question 1e

Working with Text-part I was able to union all of the Pickle files into one RDD. Upscaling this to the full dataset may encounter Stack Overflow errors; I am yet to test this. Additionally, making use of the ast Helpers module I am able to transform the data within the RDD to negate the need to use joins to piece the data together. The ast Helpers module allows a simple transformation of string to float without loss of accuracy. Once finished processing the data was again repartitioned and also saved as a series of Pickle files.

Question 1f

Applying the sample logic and modules I was able to calculate and save the TF.IDF easily. At each stage the algorithm would check to see if this stage had already been completed, if it had then it would use the pickled files if they existed for the next phase of the algorithm. If the pickle files were used this would be noted in the output file as “Using the Pre-Pickled Files”.

I chose to create hashed vectors whilst I was building the models. The reason for this was that it allowed me to dynamically change the hashsize without the need to save the TF.IDF to a pickle file for each hashsize, hence why this element is not mentioned in the stage above.

Question 2:

The table below should assist with identifying each part of the question;

Question	Task	Line Range
2a	Read the meta-data files from the meta directory as files	250-274 – XML Extracting function
2b	Read the meta-data file using a XML parsing module	276-317 – Processing the XML list
2c	Extract the subjects and the ID	287-301 – Find the most popular subject of the text data set
2d	Store the subjects lists per file as an RDD	301-315 – Saving the vectors as a list for later use.

Detailed Explanation of each Stage in the Algorithm:

Question 2b

Making use of the XML.etree module I was able to obtain the following results for the top 10 subjects in the Text-part dataset.

Rank	Subject	No. files in Subject	% Total
0	PS	162	21.745%
1	PR	111	14.899%
2	PZ	87	11.678%
3	Science fiction	69	9.262%
4	Short stories	57	7.651%
5	AP	39	5.235%
6	Fiction	22	2.953%
7	Popular literature -- Great Britain -- Periodicals	18	2.416%
8	Adventure stories	17	2.282%
9	England -- Social life and customs -- Fiction	15	2.013%

Question 3:

The table below should assist with identifying each part of the question;

Question	Task	Line Range
3a	Find the 10 most frequent subjects	292-315
3b		
I	Naive Bayes Classifier	350-370
ii	Decision Tree Classifier	371-396
iii	Logistic Regression Classifier	397-417
3c	Finding the best regularisation values for Decision Trees and Logistic Regression using cross validation and documenting the training validation and testing error for each classification method and each regularisation value.	574-627

Detailed Explanation of each Stage in the Algorithm:

Question 3a

Once the XML processing function has collected all the information from the files, the first step is to filter out the books that are not in the TF.IDF RDD. The next stage is to find the 10 most frequent themes of books in the collection. I have saved the list of each of the subjects and the books that are associated with that subject in separate pickle files.

ontr

Question 3b

The first stage of this phase is to break the hashed RDD into two subsets, training and test, in an eighty twenty split. The method of dividing up the data is dependent on creating a list; taking a random selection of that list and then using the two lists to filter the files that are or are not in the list. I found this method to be considerably quicker than using the subtract function in Spark. At this stage the binary classification is computed and also the features vector is hashed.

Question 3bi-iii

Implementation of each method is very similar; the only exception is the decision trees. This method requires the labels and features to be separated and then joined at a later stage when evaluating the method for accuracy, time to process and the F-score.

Question 3c

Spark has the function to manipulate both the Decision Trees (DT) and Logistic Regression (LR) machine learning algorithms. From testing the algorithms on Text-part it was found that the most appropriate configuration for the DT method was to use the setting shown below, testing was conducted on a local machine and not Lewes. This was found to be the maximum number of Bines and Depth that the machine was able to handle.

Impurity='gini', maxDepth=2, maxBins=3

The LR method initially yielded no usable results for the f-score, this cadicts results that should have been achieved. The expected result was that the LR method would be the most that it is the most accurate and appropriate based on the results from previous experimental results by Chi Wai Lau (2011). After experimenting with the input parameters, it was possible to tune the LR model for this text classification problem. The best solution is shown below highlighted in yellow.

Run	Hashsize	Subject	Cross Validation Fold	Model	Training Accuracy	Training F-Score	Test Accuracy	Test F-Score	Time to Evaluate Model	Number of Iterations	Minibatch Size	Step Size
1	10000	1	1	3	78.3109405	-1	78.3109405	-1	16.269279	100	0.3	0.125
5	10000	1	1	3	77.3512476	0.016666667	77.3512476	0.016666667	15.71296597	100	0.25	0.125
9	10000	1	1	3	80.61420345	-1	80.61420345	-1	15.32677603	100	0.225	0.125
13	10000	1	1	3	78.88675624	-1	78.88675624	-1	14.90298605	100	0.2	0.125
17	10000	1	1	3	74.47216891	0.310880829	74.47216891	0.310880829	15.05316305	100	0.175	0.125
21	10000	1	1	3	79.2706334	-1	79.2706334	-1	15.42590094	100	0.15	0.125
25	10000	1	1	3	78.11900192	-1	78.11900192	-1	15.39700198	100	0.1	0.125
29	10000	1	1	3	32.0537428	-1	32.0537428	-1	15.1314981	100	0.05	0.125
33	10000	1	1	3	32.0537428	0.397959184	32.0537428	0.397959184	15.06769705	100	0.025	0.125
37	10000	1	1	3	37.81190019	0.372093023	37.81190019	0.372093023	15.69438124	100	0.01	0.125

Question 4:

Question 4a

Results from Text-part are shown below:

Detailed results can be viewed in the object attached in the appendix [6]:

Time Results:

Average - Time	Data										
Model	1	2	3	4	5	6	7	8	9	10	**Total Result**
1	9.6115362474	9.6025676387	9.5304249287	9.5052359581	9.4393483775	9.508149457	9.485948801	9.4844841991	9.6121264287	9.5375260489	**9.5317348085**
2	22.989342172	23.070141271	22.903695437	22.676544217	22.663482898	22.737579244	22.807532941	22.339998123	22.777381635	22.22219318	**22.718789112**
3	14.044284538	13.886789891	13.892930698	13.812592687	13.764848665	13.796151045	13.780045325	13.812438447	13.89724162	13.825852057	**13.851317497**
Total Result	**15.548387652**	**15.519832934**	**15.442350355**	**15.331457621**	**15.289226647**	**15.347293249**	**15.357842356**	**15.212306923**	**15.428916561**	**15.195190429**	**15.367280472**

Accuracy Results:

	Data	Hashsize															
	Average - Training Accuracy							Average - Test Accuracy							**Total Average - Training Accuracy**	**Total Average - Test Accuracy**
Model	500	1000	2500	5000	7500	10000	15000	500	1000	2500	5000	7500	10000	15000		
1	92.3104	92.9279	93.6359	93.4698	94.2668	92.5050	94.3238	92.3104	92.9279	93.6359	93.4698	94.2668	92.5050	94.3238	**93.3485**	**93.3485**
2	92.4463	92.6661	98.3389	98.8322	93.2383	99.0134	93.8691	92.4463	92.6661	98.3389	98.8322	93.2383	99.0134	93.8691	**95.4863**	**95.4863**
3	91.9581	91.9950	92.0336	92.0403	91.9362	91.9866	92.0017	91.9581	91.9950	92.0336	92.0403	91.9362	91.9866	92.0017	**91.9930**	**91.9930**
Total Result	**92.2383**	**92.5296**	**94.6695**	**94.7808**	**93.1471**	**94.5017**	**93.3982**	**92.2383**	**92.5296**	**94.6695**	**94.7808**	**93.1471**	**94.5017**	**93.3982**	**93.6093**	**93.6093**

F-Score Results:

Where the value is either negative or -1 this suggests that the value of the F-Score was not obtained. This usually happened when the denominator of the f-score was equal to zero.

	Data	Hashsize															
	Average - Training F-Score							Average - Test F-Score							**Total Average - Training F-Score**	**Total Average - Test F-Score**
Model	500	1000	2500	5000	7500	10000	15000	500	1000	2500	5000	7500	10000	15000		
1	-0.6288	-0.3441	-0.3801	-0.2260	-0.5883	-0.0979	-0.3969	-0.6288	-0.3441	-0.3801	-0.2260	-0.5883	-0.0979	-0.3969	**-0.3803**	**-0.3803**
2	-0.4674	-0.4877	-0.3656	-0.3354	-0.2515	-0.3696	-0.0465	-0.4674	-0.4877	-0.3656	-0.3354	-0.2515	-0.3696	-0.0465	**-0.3320**	**-0.3320**
3	-1.0000	-1.0000	-1.0000	-1.0000	-1.0000	-1.0000	-1.0000	-1.0000	-1.0000	-1.0000	-1.0000	-1.0000	-1.0000	-1.0000	**-1.0000**	**-1.0000**
Total Result	**-0.6988**	**-0.6106**	**-0.5819**	**-0.5205**	**-0.6133**	**-0.4892**	**-0.4812**	**-0.6988**	**-0.6106**	**-0.5819**	**-0.5205**	**-0.6133**	**-0.4892**	**-0.4812**	**-0.5708**	**-0.5708**

Question 4b

Results for Text-Part

Naive Bayes for experimentation was shown to be the fastest method however, when taking into consideration the F-Score the best method would be the Decision Tree model. On average the accuracy and F-Score values concluded greater consistently for all results. The logistic regression model did not yield any results that could be used for comparison, all of the F-Score values on average were equal to minus one indicating that an error had occurred in the calculation of either the recall or precision variable. This was peculiar since the LR optimisation values have been test and run on the hashed vector size of 10000. In conclusion, with the dataset used the most appropriate method for building a text classification system would be the DT method.

Results for Text-Full

Calculation of the Word Frequency vectors processed fully without any issues, however when calculating the IDF an error ensued. The issue resided with Lewes based on the error output and was a found to be a limit on the number of open files that I was able to process. The hard and soft limitation of my user profile is 16384 files, in comparison the system wide maximum limit is 13118730 files. I concluded that if I had run Text-Full as the root user it would have likely not impacted the calculation of both the IDF and TF.IDF RDD's. Please refer to appendix [8] for further details.

Naive Bayes

Naive Bayes (NB) is a probabilistic learning method that can be efficiently applied to a many domains as a classification method (both binary and multinomial classification is possible in Spark). NB is a generative classification method whereby the method is dependent on the joint likelihoods of the features. This condition is

expected, however in most cases of text classification this expectation of the model is not actually achieved. As seen in the summarised results table, the model is fast to build and test in comparison to the other machine learning algorithms that are used in this coursework.
(NB can be thought of as Linear in complexity)

Decision Trees

Decision Tree (DT) Analysis is exponential in complexity when building the model but when testing the model complexity is linear, this characteristic can be seen in the summarised table. If the depth and width of each of the nodes are increased it would be expected that the time to build the tree would increase exponentially with the depth of the tree as it identifies pertinent features in the vector. The algorithm is a deterministic classifier, one advantage of this classifier over other stochastic methods is that the tree is interpretable to a human, therefore validation is easier. The disadvantage of this method is that there are tuning parameters that determine the depth of the tree. Caution needs to be taken when deciding the initial parameters, a complex model that fits the training data with minimal error is likely to be a poor classifier when applied to a test dataset. Another issue with a decision trees is that they suffer from fragmentation; this is most often caused by high dimensionality where many features may produce a decision tree that is too large for use and interpretation. Nevertheless, the use of a hashed feature vector can be used to avoid this occurring.

Logistic Regression

Logistic Regression (LR) is a discriminative statistical classifier that utilises the MLCE (or MAP with regularisation) to fit a probability model. Logistic Regression doesn't suffer from the curse of dimensionality in the same way the Decision Tree method does, except when attempting to validate the results. It does however suffer for optimisation issues that mean it requires the users to tune (adjust) to find the most optimised values for the input parameters; step size, number of iterations, regularisation method and sampling size. The method to Logistic Regression that Spark uses is through the use of Stochastic Gradient Decent, careful consideration has been to be made to determine which parameter will increase the accuracy. Tuning of the parameters can be seen in the results of 3c, it would be ideal if there was an additional in-built algorithm that could be utilised to numerical optimise the input parameters. Using the current method it is hard to know if either the global maximum or a local minimum have been found. Visually it's not possible to verify if the algorithm has found one of the maxima, given that the feature vectors are in the orders of thousands of dimensions.

Question 4c
Google's self-driving Vehicles:

Road layouts, and particularly infrastructure, is constantly changing, therefore it is not feasible for Google to collect all the possible spatial data required for navigation. The application and design of algorithms to run self-driving vehicles are capable of distinguishing between different objects, both stationary and non-stationary when navigating along a road.

Obstacle detection, autonomous navigation and the computational awareness of road driving standards will all be implemented through multiple machine learning algorithms (supervised and unsupervised) that are capable of "active learning" on the spot able to deal with a situation when it arises.
Extensions:

- The inclusion of meta-data that can be used to enhance the models; possible data could include traffic updates and weather data.
- The development of a deep learning algorithm techniques

Optimisation:

- Add more data from related sources (keep simple models and enrich them)
- The development of a deep learning algorithm as a replacement of a more sophisticated image recognition classification system.

Speech Recognition:

Automatic speech recognition (ASR) is the development of algorithms that are capable of changing text to speech and vice versa. ASR has been applied in many industries including search engines (Google), mobile personal assistants (Siri - Apple), telephony (Nuance Voice Control) and language learning software (Rosetta Stone). These

systems usually use statistical models and speech can be considered as a stochastic model. Treating the problem as a stochastic model means that it is computationally feasible to determine context and phoneme of words or subsets of words.

Extensions:

- The application of other natural language processing (Stemming, Topic Segmentation, Auto-Summation, Chucking techniques and disambiguation of words)
- Spam detection, genre and topic identification and malicious submission of reviews.

Optimisation:

- Performance tuning of digital signal processing (ie. Voice to Digital Signal)
- Feedback loops where the un-seen test data can be used to optimise the current models.

Task for Pairs (Extra Work) - *Completed Independently:*

Extracting the Author's Year of Birth:

Using the function below this was possible to extract the data into a list.

```
def xmlExtractDate(files, table):
    if '.rdf' in files:
    # Import into the XML Parser
    root = ET.parse(files).getroot()
    # Getting the Book ID
    ebookChild = root.find('{http://www.gutenberg.org/2009/pgterms/}ebook')
    book = ebookChild.get('{http://www.w3.org/1999/02/22-rdf-syntax-ns#}about')
    rgID = re.compile('.*?(\\d+)', re.IGNORECASE|re.DOTALL)
    ebookID = rgID.search(book).group(1)
    try:
    # Obtaining the Birth Date
        creatorDetails = ebookChild.find('{http://purl.org/dc/terms/}creator')
        creatorDetailsAgent = creatorDetails.find('{http://www.gutenberg.org/2009/pgterms/}agent')
        Date = creatorDetailsAgent.find('{http://www.gutenberg.org/2009/pgterms/}deathdate')
        table.append([ebookID, Date.text])
    except AttributeError:
        x = 1
return table
```

Train a linear regression model (LassoWithSGD from mllib) on the TF.IDF vectors.

How this was implemented into the algorithm can be found in the object below:

Prediction_Birthdate.py

Results:

LassoWithSGD.xlsx

Discussion:

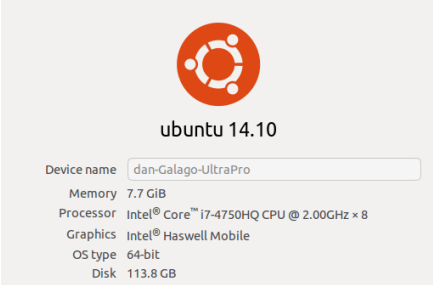
From inspection of the results it can be seen that there is a significant difference (918 on average) between the actual label and the predicted value from the trained model. The reason for this is that LassoWithSGD shares the same underlying method of finding the maxima as the LR method in Spark does. Therefore in order to achieve good results it will require tuning to find the optimised values for the input parameters. I would expect that even after improvement (optimisation) that this would only give an approximation of the birth date to within a certain degree of accuracy.

References:

Chi Wai Lau. (2011). News Recommendation System Using Logistic Regression and Naive Bayes Classifiers. Available: http://cs229.stanford.edu/proj2011/Lau-NewsRecommendationSystemUsingLogisticRegressionAndNaiveBayesClassifiers.pdf. Last accessed 4th Dec 2014.

Appendix:

[1]- Location Machine Specification:



[2] - Algorithm – ackf415-new.py:


ackf415-new.py

[3]- Algorithm – ackf415-lewes.py:


ackf415-lewes.py

[4] - Algorithm – ackf415-Text-Full.py:


ackf415-lewes-textfull.py

[5] - Top 10 Subjects:


Top_10_Subjects.csv

[6] - Detailed Results Spreadsheet:


Extracting_Results.xlsx

[7] - Coursework Sheet:

This is the coursework that I followed, version 1.

Adobe Acrobat Document

[8] - Lewes Open File Limit Commands:

A quick analysis of the limits provided these results.

```
ackf415@lewes:~$ cat /proc/sys/fs/file-max
13118730
ackf415@lewes:~$ ulimit -Hn
16384
ackf415@lewes:~$ ulimit -Sn
16384
ackf415@lewes:~$
[textparta 0:su*
```

[9] – Algorithm – ackf415-new-LRoptimisation.py

ackf415-new-LRoptimisation.py

[10] – Logistic Regression Optimisation Results

LR_Optimisation.csv

[11] – Lewes – Output File for Text-Part

output.txt

[12] – Lewes – Output File for Text-Full

output_TF.txt